Project and Data Analysis

Data Characteristics

1. **Nature of Data**:
    - The data represents product information, with each JSON document describing a specific product.
    - Attributes include:
        - Product ID, Product Name, Brand, Description, Attributes, etc.
2. **Data Challenges**:
    - **Discrepancies**: Missing or inconsistently formatted fields.
    - As Seen below different Values for Price field in different formats

      "Price": "$999","Price": 90,"Price": "one hundred fifty"

      "Quantity": 25, "Quantity": null, "Quantity": "",

    - **Non-uniformity**: Varying attributes across payloads:
        - Example:
        - Each Payload below has different Attribute fields mentioned inside it

      Payload 1:
      json
      Copy code

```
"Attributes": { "Size": "9-inch",
"Frozen": true,
"Servings": "2-3" }
```

        -

      Payload 2:
      json
      Copy code

```
"Attributes": { "Weight": "4 oz",
"Packaging": "Wrapper",
"Refrigerated": false }
```

        -

    - **HTML Tags and Date Formatting**:
        - Presence of HTML and Hash tags in fields like "data".

          "data": " <h1>iPhone 13 Pro</h1><p>Experience the future with the new iPhone 13 Pro in Sierra Blue! Featuring a Super Retina XDR display and A15 Bionic chip. Now with 5G speed for lightning-fast downloads.

Storage options: 128GB, 256GB, 512GB, and 1TB. Limited warranty of 1 year included. Key features: Triple-camera system, Super Retina XDR display, 5G connectivity. Don't miss out! "

"data": "## Air Force One Sneakers\n\n**Brand:** Nike\n\nStep into the iconic style of the Air Force 1 Sneakers! Available in sizes 7-11. Features include:\n- Leather Upper\n- Rubber outsole\n- Perforations for ventilation\n\n*Color options: WHITE only.* Perfect for everyone - unisex design."

- Inconsistent date formats across payloads.

```
"DateAdded": "2021/09/24"
"DateAdded": "24-09-2021"
"DateAdded": "September 24, 2021"
```

Impact of Challenges

- Hard to analyze trends and use the data for machine learning (ML) model training.
- Missing attributes and discrepancies reduce the data's reliability and usability.

Data Cleaning Solution

A Python-based solution is proposed to:

1. **Standardize Data**:
    - Resolve typos and inconsistent formatting.
    - Address missing values.
2. **Uniform Formatting**:
    - Remove HTML tags.
    - Standardize date formats.

Understanding Use Cases and Query Patterns

Goals for Optimization

1. **Faster Inserts**:
    - Minimize indexing overhead during data ingestion to maintain high throughput.
    - Avoid over-indexing, which can slow down write operations.
2. **Faster Reads**:
    - Identify common query patterns to optimize filtering and sorting criteria.
    - Design indexes to improve query performance for frequent access patterns.
3. **Balanced Workload**:
    - Maintain a balance between frequent writes (inserts/updates) and reads.

Common Query Patterns

1. **Retrieve Product Details by ID**:
    - Use case: Fetch specific product information using `ProductID`.
2. **Filter by Category or Brand**:
    - Use case: Retrieve grouped data, e.g., "Find all products in the Electronics category."
3. **Range Queries on Numeric or Date Fields**:
    - Use case: Search for products within a price range or added during specific date ranges.
4. **Text Searches in Description or Data Fields**:
    - Use case: Perform text-based queries to find products containing specific keywords in their description or data fields.

Design Considerations for Optimization

- Prioritize indexing on fields frequently used in queries:
    - `ProductID` for lookups.
    - `Category` and `Brand` for grouped reads.
    - `Price` and `DateAdded` for range queries.
    - Full-text search indexes for text-based queries (if supported by the database).
- Use compound indexes for combined filtering (e.g., `Category` + `Brand` queries).

Feature Store Setup: MongoDB

Why MongoDB?

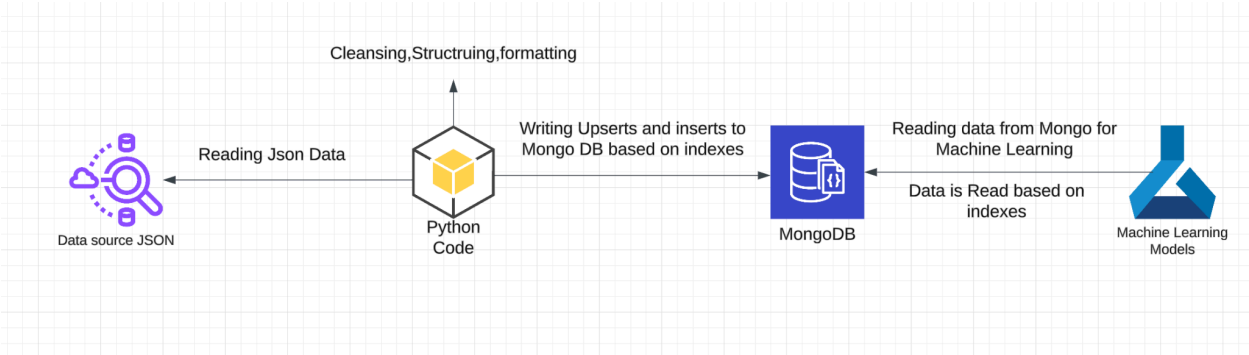
1. **Document-Based Storage**:

    - JSON-like format ensures compatibility with incoming data.
2. **Optimized for Read Operations**:
    - Fast data retrieval with appropriate indexing.
3. **Scalability**:
    - Horizontal scaling via sharding.
4. **Query Routers**:
    - Efficient routing of client requests to appropriate shards.

Indexing Strategy

1. **Primary Index**: `_id` (default MongoDB index).
2. **Single-Field Indexes**:
    - `ProductID`: Fast lookups.
    - `Category`: Grouped reads (e.g., finding electronics).
    - `Brand`: Filtering by brand.
    - `Price` and `DateAdded`: Range queries and sorting.
3. **Compound Indexes**:
    - `Category_1_Brand_1`: Filtering by category and brand.
    - `Price_1_DateAdded_-1`: Efficient range queries and sorting by date.

Current Solution Architecture



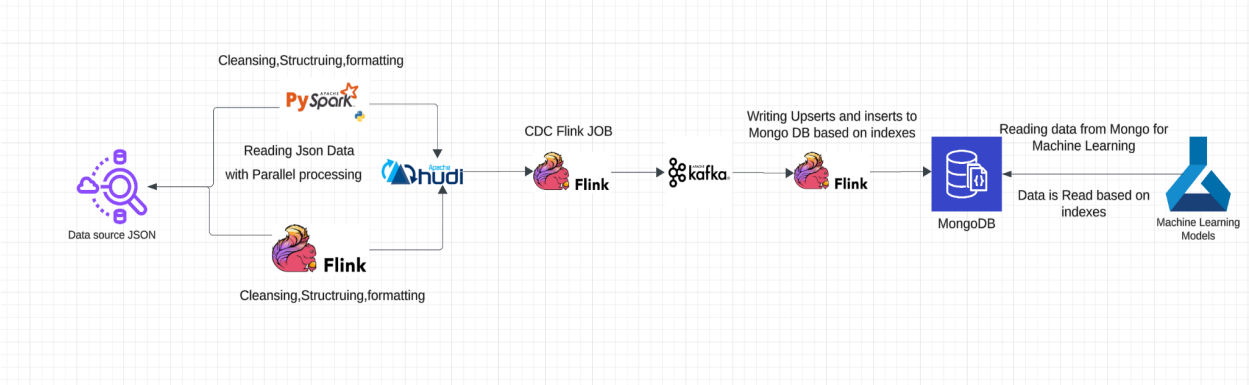
Workflow:

1. **Data Ingestion**:
    - Read JSON from source.
2. **Processing**:
    - Clean and standardize data.
3. **Storage**:
    - Insert cleaned data into MongoDB.

Challenges:

1. **Limited Scalability**:
    - Single-point data ingestion may struggle with large JSON files.
2. **Delta Management**:
    - Handling incremental updates or bulk upserts efficiently.

Proposed New Architecture (Detailed)



The new architecture is designed to improve scalability, real-time data processing, and efficient data storage, while addressing the limitations of the current architecture. By leveraging distributed systems and decoupling the storage and processing layers, the architecture ensures better performance and flexibility for handling large and continuously changing datasets.

1. Data Processing: Distributed Framework

The key change in the new architecture is the use of distributed processing frameworks like **Apache Flink** and **PySpark**. These technologies allow the system to process large volumes of data in parallel, making it more scalable and efficient compared to the current single-threaded processing approach.

- **Apache Flink**: Flink is a stream processing framework designed for real-time data processing. It allows us to process data in near real-time with low latency. Flink is particularly suitable for incremental processing and handling Change Data Capture (CDC)
- **PySpark**: PySpark is the Python interface for Apache Spark, a distributed data processing engine. It can handle both batch and real-time data processing, making it suitable for large-scale ETL (Extract, Transform, Load) operations. PySpark will be used for preprocessing the data (cleaning, transforming, etc.) before storing it in the data lake.

By using Flink and PySpark, we can handle larger data volumes, improve processing speed, and ensure that data is processed in parallel, reducing bottlenecks.

---

2. Data Storage: Apache Hudi as the Data Lake

Instead of directly writing the processed data into MongoDB, the data will first be stored in a **Data Lake** managed by **Apache Hudi**. Hudi provides several key advantages for handling large-scale datasets:

- **Versioned Storage**: Apache Hudi allows us to store data in a versioned format. This means we can keep track of historical data and even "time travel" to a previous version of the data if needed. This is especially useful for audit trails and rollback scenarios.

- **Efficient Storage Formats**: Hudi stores data in **Parquet format**, which is highly efficient for both storage and processing. Parquet's columnar storage format optimizes read and write operations, significantly reducing storage costs while improving performance.

- **Incremental Processing**: Hudi supports incremental data processing and Change Data Capture (CDC), which is ideal for sending only the changed data to downstream systems like MongoDB. Instead of processing all the data every time, Hudi allows for processing only new or updated records.

- **Compression and Optimization**: Hudi supports data compression, which helps reduce storage overhead. It also provides indexing techniques like **Bloom filters** that improve query performance by reducing the amount of data scanned during reads.

- **SQL Query Interface**: With Hudi, we can query the data directly using SQL, making it easier for teams to access the data without needing to rely on MongoDB for complex queries. This is beneficial for analytical use cases, where large-scale queries need to be run without overburdening the primary MongoDB database.

---

3. Real-Time Data Propagation: Change Data Capture (CDC)

Once the data is processed and stored in Hudi, we need to ensure that any changes (inserts, updates, or deletions) are propagated to MongoDB in near real-time. This is where **Change Data Capture (CDC)** comes into play.

- **Flink CDC**: Flink will be used to read the incremental changes (CDC) from Apache Hudi. Flink's CDC connector allows us to capture changes in real time, ensuring that only the updated data is processed. These changes will be captured as events and sent to **Kafka**, which serves as a message broker to handle real-time data streams.

- **Kafka**: Kafka will act as an intermediate layer between Flink and MongoDB. As an event-driven messaging platform, Kafka will ensure that data is sent in real time to downstream systems like MongoDB without introducing bottlenecks. Kafka will manage the high throughput and stream the changes efficiently.

- **Kafka:** In case of high data Volume we can also increase the partition of kafka for a more better throughput .

- **FLINK:** Here FLINK can be used to read data from parallel kafka partition and process them in a parallel manner.we can also increase the Task Manager memory provided to flink in case there is high traffic of data

---

4. Real-Time Upsert Logic: Flink to MongoDB

After the CDC events are sent to Kafka, another **Flink job** will consume these events. The job will apply the **upsert logic** to MongoDB, ensuring that new or updated records are inserted or updated correctly.

- **Upsert Logic**: Upsert operations allow us to insert new data and update existing records in MongoDB without duplicating data. By using the unique product identifier (`ProductID`), we can efficiently determine whether a record already exists and whether it needs to be updated.

This part of the architecture ensures that MongoDB always contains the latest product data, with updates applied in real-time.

Advantages of the New Architecture

1. Scalability

- **Apache Hudi**: Efficient storage with versioning and incremental processing capabilities enables us to handle large datasets without compromising performance.

- **Kafka and Flink**: Both Kafka and Flink are designed to scale horizontally, allowing the architecture to handle high throughput and real-time data streams. This scalability ensures that the system can handle growing data volumes and increasing query loads as the application grows.

2. Real-Time Capabilities

- By leveraging **Flink's real-time processing** and **Change Data Capture** (CDC), the architecture enables near real-time data processing. This ensures that MongoDB is always updated with the latest product data without delays, making the system highly responsive.

3. Data Versioning

- **Apache Hudi** provides **data versioning**, which enables rollback, auditing, and time-travel queries.

4. Decoupled Architecture

- The new architecture decouples the data processing, storage, and ingestion stages. This separation of concerns makes the system more modular, easier to maintain, and scalable. It also allows different teams to consume the data in different formats (e.g., directly from Hudi or MongoDB) depending on their needs.

5. Flexibility

- With Kafka, we gain the flexibility to add multiple consumers downstream. These consumers could be used for different use cases such as real-time dashboards, machine learning models, or additional analytics services. Kafka's ability to handle multiple consumers makes the architecture adaptable to future requirements.

Trade-Offs and Challenges

1. Complexity

- The new architecture involves multiple components (Hudi, Flink, Kafka, MongoDB), which increases system complexity. Each component requires careful configuration, monitoring, and maintenance to ensure smooth operation.

2. Operational Overhead

- With the addition of distributed systems, there is a need for increased monitoring and management. Components like Flink and Kafka require dedicated infrastructure and expertise, which may add operational costs.

3. Cost

- Distributed systems like Kafka, Flink, and Hudi incur additional infrastructure costs compared to a simpler MongoDB-based solution. However, these costs are justified by the improved scalability, real-time capabilities, and flexibility of the system.

Comparison of Scenarios

Feature	**Current Architecture**	**New Architecture**
Data Ingestion	Direct to MongoDB	Staged via Apache Hudi
Processing	Single-threaded Python processing	Distributed with PySpark/Flink
Scalability	Limited	High
Real-Time Support	Batch mode only	Real-time CDC with Flink
Data Versioning	Not available	Available with Apache Hud
Flexibility	Limited to MongoDB consumers	Supports diverse downstream use cases

---

Conclusion

The new architecture offers significant improvements in scalability, real-time processing, and flexibility by integrating distributed data processing tools like Flink, Kafka, and Hudi. It is designed to handle large datasets with incremental changes, allowing for efficient data storage, real-time updates, and powerful query capabilities. While it introduces complexity and additional costs, the long-term benefits in terms of performance, flexibility, and scalability make this approach a robust solution for handling the growing data needs of the system.

Final Data Output:


```
_id: 1
Allergens : null
▸ Attributes : Object
BatteryLife : null
Brand : "Apple"
Category : "Electronics"
Color : "Sierra Blue"
Count : null
Coverage : null
DateAdded : 1632441600000
Description : "The iPhone 13 Pro represents Apple's latest leap forward in smartphone…"
DescriptionLength : 179
Flavor : null
Frozen : null
Gender : null
GlutenFree : null
Hypoallergenic : null
▸ KeyFeatures : Array (3)
Organic : null
Packaging : null
PetSafe : null
Power : null
Price : 999
ProductName : "iPhone 13 Pro "
Quantity : 50
RAM : null
Refrigerated : null
Season : null
Servings : null
Size : null
Storage : "128GB"
Voltage : null
Volume : null
Warranty : "1 year limited warranty"
Weight : null
data : "iPhone 13 ProExperience the future with the new iPhone 13 Pro in Sierr…"
▸ Hide 11 fields
```

```
_id: 3
Allergens : null
Attributes : Object
BatteryLife : "2 weeks"
Brand : "Phillips"
Category : "Home"
Color : "Black"
Count : null
Coverage : null
DateAdded : 1632441600000
Description : "Experience superior oral health with the Sonicare DiamondClean Toothbr…"
DescriptionLength : 181
Flavor : null
Frozen : null
Gender : null
GlutenFree : null
Hypoallergenic : null
KeyFeatures : Array (3)
  0: "Five brushing modes"
  1: "USB charging travel case"
  2: "Improves gum health"
Organic : null
Packaging : null
PetSafe : null
Power : null
Price : 150
ProductName : "Sonicare DiamondClean Toothbrush"
Quantity : 25
RAM : null
Refrigerated : null
Season : null
Servings : null
Size : null
Storage : null
Voltage : "110-220V"
Volume : null
Warranty : null
Weight : null
data : "Sonicare DiamondClean Toothbrush by PhillipsColor: BlackVoltage: 110-2…"
```


25